Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 403 (2008) 133–159
Theoretical Computer Science
www.elsevier.com/locate/tcs

Polyhedra genus theorem and Euler formula: A hypermap-formalized intuitionistic proof☆

Jean-François Dufourd*

Université de Strasbourg UFR de Mathématique et d'Informatique, Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection (LSIIT, UMR CNRS-ULP 7005), Pôle API, Boulevard Sébastien Brant, 67400 Illkirch, France

Received 1 August 2006; received in revised form 4 February 2008; accepted 5 February 2008

Communicated by D. Sannella

Abstract

This article presents formalized intuitionistic proofs for the polyhedra genus theorem, the Euler formula and a sufficient condition of planarity. They are based on a hypermap model for polyhedra and on formal specifications in the Calculus of Inductive Constructions. First, a type of free maps is inductively defined from three atomic constructors. Next, a hierarchy of types defined by invariants, with operations constrained by preconditions, is built on the free maps: hypermaps, orientated combinatorial maps and a central notion of quasi-hypermaps. Besides, the proofs of their properties are established until the genus theorem and the Euler formula, mainly using a simple induction principle based on the free map term algebra. Finally, a constructive sufficient condition for polyhedra to be planar is set and proved. The whole process is assisted by the interactive Coq proof system.

Keywords: Polyhedra; Hypermaps; Genus; Euler formula; Formal specifications; Computer-aided proofs; Coq system

1. Introduction

This article presents formal specifications and computer-aided proofs in the area of topological modelling. It mainly deals with a question debated since the 18th century, namely giving a rigorous proof of the *Euler formula*: $v - e + f = 2$, where v, e and f respectively are the numbers of vertices, edges and faces of a connected polyhedron. For instance, a cube has 8 vertices, 12 edges and 6 faces: $8 - 12 + 6 = 2$.

Seemingly innocent, this problem has in fact for a long time illustrated the reflexion and the controversy about what polyhedra and proofs of their characteristics really are. Numerous propositions have been made during the last three centuries, from the most intuitive to the cleverest, to precise the polyhedron concept and to prove the formula.

It is now widely recognized that such a proof is more convincing if geometric arguments are avoided in favour of topological ones, or, even better, purely combinatorial ones. Indeed, combinatorial arguments are rather primitive and

☆ This work is supported by the French ANR through the "white" project GALAPAGOS (2007–2010).
* Tel.: +33 03 90 24 45 55.
E-mail address: dufourd@dpt-info.u-strasbg.fr.

0304-3975/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2008.02.012

easier to deal with than those coming from a subtle axiomatics of geometry. Correctly addressing such issues is a good reason for the interest of mathematicians in combinatorial models of polyhedra.

These models are also crucial for the computer scientists who deal with geometrical *modellers*, i.e. systems for building, updating, traversing and visualizing geometrical objects [41,33,4]. They also made numerous propositions to represent polyhedra, more or less close to implementations. It now appears that the consensus between the two communities can be found in formalized discrete models. One of the best is the *combinatorial oriented map*, a kind of functional multigraph, which allows us to describe the combinatorial topology of an important class of polyhedra [45]. In addition to being a precise algebraic model, it is easily extended into an attractive homogeneous concept of *hypermap* [8].

The notion of hypermap and the operations it involves can be axiomatized to prove properties of polyhedra formally. One of the most significant is probably the *genus theorem* which states that, for a polyhedron embeddable on an orientatable closed surface − or the corresponding combinatorial orientated map −, the Euler characteristic $\chi = v - e + f$ is always an even integer, possibly negative, and that, c being the number of connected components of the polyhedron, its genus $g = c - \chi/2$ is a natural number. In fact, g corresponds to the number of holes in the surface of the polyhedron. When $g = 0$, the surface is without holes, and the polyhedron is said to be *planar*. When it is also *connected* ($c = 0$), it satisfies the Euler formula. Actually, the real problem is to find conditions under which the planarity of polyhedra is ensured.

Thus, in this paper, we present a purely combinatorial intuitionistic proof of the genus theorem and of a sufficient condition for polyhedra to be of genus 0 − i.e. to satisfy the Euler formula − based on the hypermap structure. The hypermap framework is entirely formalized and the proof is developed interactively and verified by a proof assistant. This requires the introduction of a sizeable amount of notions and the proof of numerous lemmas which are profitably computer-aided. Furthermore, formal proofs are becoming crucial in a controversial field such as ours. We have opted for a formalization in the *Calculus of Inductive Constructions* and the use of the *Coq system* developed at INRIA [34,6,24,3].

Among the authors who stand out in the field, Jacques and Tutte pioneered the combinatorial map area in the 1960s, and proved the Euler formula [25,44,45]. Recently, Bauer and Nipkow presented a proof assisted by Isabelle/Isar of the Euler formula for triangulations as a step in the formalized proof of the five-colour theorem [1]. Finally, in 2005, Gonthier achieved to formalize and to prove a great result, namely the four-colour theorem, using Coq [21]. As he says, hypermaps with ad hoc operations, as well as planarity and the Euler formula, played a big role in his development.

Compared to these approaches, ours appears straightforward, because it does not presuppose any particular knowledge about topology, geometry, or even combinatorics. It is entirely based on a free algebra of hypermap terms, which is built from atomic constructors. To perform the proofs, it mainly uses a simple induction principle on these terms. The proofs are fully intuitionistic, i.e. they never use the law of the *excluded middle* (nor any equivalent). In addition, our framework allows to recover by composition notions and operations appearing in the other approaches. The price to be paid is a careful construction of a hierarchy of types defined by invariants and operations constrained by preconditions: free maps, quasi-hypermaps and hypermaps. Actually, this construction is similar to the one we have used to design and develop geometrical software by algebraic specifications. In a sense, the present work on the Euler formula appears as a step for investigations concerning the proofs of topological and geometrical map-based algorithms.

The main features of the Coq language and system used in our specifications and proofs are reminded in the following sections. The whole process is described and explained, but the full details of the proofs are out of the scope of this article. Note that the complete development is included in the Coq users'contributions and fully accessible [16]. The rest of the paper is organized as follows. In Section 2, we summarize related work concerning the Euler formula, geometrical modelling, and computer-aided proofs, particularly in topology and geometry. In Section 3, we recall some basic mathematical definitions and properties about polyhedra, hypermaps, cells of hypermaps, genus, Euler characteristic, planarity and embeddings. In Section 4, we present some preliminary notions and we inductively define a type of free maps, in which all our specifications are rooted. Then, we investigate the central notion of quasi-hypermap, a kind of open hypermap, and we specify hypermaps and the closure of a quasi-hypermap into a hypermap. In Section 5, we define faces and paths in faces. Finally, we inductively specify characteristics, and prove the genus theorem, the Euler formula and characterize the planarity. Section 6 presents some concluding remarks and outlines future work.

2. Related work

2.1. Euler formula

It seems that L. Euler discovered the formula which bears his name in 1752, and that A.M. Legendre gave the first proof in 1794. Several authors tackled the higher-dimensional case of polytopes in the 19th century. It is admitted that the first real general proof was elaborated by H. Poincaré in 1899, using homology and linear algebra [9,32]. An interesting bibliographic study and numerous pieces of information about the Euler formula are summarized in a web site administrated by Eppstein, *The Geometry Junkyard* [18]. The site currently presents a list of nineteen proof sketches of Euler's formula. Some of them use the geometry of the polyhedron, others the topology of the underlying graph, and others still the geometry of the graph embedding. They implement different proof strategies, the most common being induction on vertices, edges or faces.

However, without a rigorous axiomatic framework, it is difficult to be convinced by the accuracy of some arguments used in the proofs. Let us give an example stemming from an induction driven by the edges to prove the Euler formula. When, during the proof, an edge is added between two existing vertices, one often admits that a face is cut in two, which preserves the relationship $v-e+f=2$. This is true if the pohyhedron subdivides an *orientatable closed* surface − which excludes for instance the Klein bottle and the projective plane [22] −, and if the edge is added *within* a face.

To fulfil the first requirement, the choice of the combinatorial model of orientatable closed surfaces is of prime importance. In fact, many models are not precise and formal enough to be sure that they describe the relevant field. We must also ensure the second property, namely edge addition within a face. Then, we need to know what faces separated by an edge really are, a property related to the Jordan curve theorem. Roughly speaking, this theorem asserts that, on the sphere (or on the plane), the complementary of a curve homeomorphic to a circle (a Jordan curve) is open and disconnected [32]. However, this theorem, at least in a discrete version, can be proved from the Euler formula, as Tutte does [44]. As a result, we face a not inconsiderable risk of vicious circle: proving Euler's formula requires the Jordan curve theorem, which is proved using Euler's formula.

Therefore, in addition to its fascinating beauty and apparent simplicity, the Euler formula served several times as a major example of epistemological investigations of what a property, as well as its discovery and its proof, really are. The work of Lakatos is amongst the best known in this respect [29].

2.2. Models of polyhedra

In computer science, models of polyhedra have been investigated in many ways depending on what operations are privileged [41]. The so called *Boundary Representation* describes the objects by their boundaries in all dimensions, i.e. considers them as subdivisions in cells: vertices, edges, faces, volumes, etc. In this representation, a particular importance is given to incidence and adjacency relationships between cells. The shape and exact location of the cells in space are considered in a second step only. This is the vision we adopted for many years.

To model subdivisions of surfaces − or polyhedra − in cells, various propositions have been made. The *winged-edge* is a low-level representation of edges as structures, containing pointers to incident vertices and faces and to adjacent edges [2]. It is widely used to implement polyhedra in geometrical modellers, with the so called *Euler operators* [33], which allow to add or remove vertices, edges or faces, while preserving the Euler formula, or even the genus. It enables us to represent subdivisions of closed and orientatable surfaces.

More abstract concepts have been imagined, mainly based on graphs, to fully capture topological properties and improve subsequent implementations, rather than dealing with pointers directly. A summary is given in [31]. Among them, one of the most studied and used is the *combinatorial oriented map*. It has the same modelling power as the winged-edge, i.e. the subdivisions of closed and orientatable surfaces, but, thanks to its algebraic formulation, it helps to rigorously define operations and extensions. The idea of using a *map* to represent a polyhedral subdivision is rather common in mathematics, but not always in its orientated form [9].

The roots of the combinatorial orientated map concept are to be found in the work by Brahana [5], in a brief summary of Edmonds [17], and in a rather substantial presentation of Jacques [26]. This notion has been extensively studied by numerous authors, e.g. W.E. Tutte in [45], and extended in different ways, resulting in the *hypermap* concept by Cori [8] or the *generalized map* concept by Lienhardt [31]. Each of these map models is exactly adapted to a class of topological varieties: open or closed, orientatable or not, in various dimensions [31].

The combinatorial orientated maps have been extensively used to build software, such as drawing tools [20]. Planar maps are used in the computational geometrical library CGAL [19]. At Strasbourg University, our most significant experiment was the development of Topofil, a modeller based on generalized maps, able to interactively manage 3D volumic subdivisions, while controlling their topological invariants. An interesting particularity is that the development of Topofil was entirely based on algebraic specifications [12,4].

2.3. Formal specifications and proofs

This algebraic specification experiment about Topofil was not completely satisfactory because some properties and operations remained doubtful. Therefore we have turned towards proof systems, while keeping our approach of formal description. We have chosen as logical support the *Calculus of Inductive Constructions*, or CiC [6,34], which is a high-order intuitionistic logic based on type theory, λ-calculus and induction. Proofs are seen as typed lambda-terms according to the Curry–Howard isomorphism. A main advantage of the CiC is that it is implemented in the Coq proof assistant [34,6,24]. The specification language Gallina, the system libraries [7] and the tactics have provided an appropriate support for all our studies. For a first glimpse into the Coq system, the reader can follow the on-line tutorial [24]. All the system features are detailed in [7]. For a comprehensive substantial Coq presentation, more orientated towards program certification, the reader can refer to [3].

Few experiments have been led in geometrical proofs aided by a proof assistant. We can mention work in basic geometry by von Plato and Kahn for an intuitionistic axiomatic [36,27] and in computational geometry by Bertot and Pichardie [35] for convex hull algorithms, both with Coq. One of the prominent results in topology is a proof with the HOL Light system of the difficult Jordan curve theorem by Hales [23] for planar rectangular grids, following the Kuratowski characterization of planarity. Finally, to our knowledge, we have done in Strasbourg the only experiments in geometrical modelling. They provided us a first version of map specifications and proofs in Coq for a condition of planarity where the genus theorem was only mentioned as a conjecture and not proved [39,38,40]. The present paper gives the proof and precises the notion of planarity, after an entire revision of the foundations of the specifications with hypermaps, taking into account the improvements of Coq. We have also conducted extensive studies to model combinatorial surfaces of any kind with generalized maps and prove the famous theorem of surface classification [22, 10,11]. However, this work does not provide the proofs we are interested in either. Finally, we recently specified and certified in Coq an image segmentation program with hypermaps [15].

2.4. Proofs for hypermaps or planar graphs

A hypermap is a finite set equipped with two permutations, also called substitutions. When one of the permutations is involutive, it is a combinatorial orientated map. In the work by Jacques et al. [25,26], operations to construct or update hypermaps are described in terms of transpositions, i.e. exchanges in permutations. The proof of the genus theorem for hypermaps is given, probably for the first time, in [25]. Using extensively known properties of the *symmetrical group* − roughly speaking the group of the permutations on a finite set − and a nonimmediate lemma of Serret, the proof follows an induction on the number of fixpoints of the substitutions. For us, the considerations at stake are not elementary enough to be completely convincing. Moreover, they are far from our previous algebraic map specifications and future needs.

A formalization in Isabelle/Isar of planar graphs in the way of Yamamoto et al. [46] is presented by Bauer and Nipkow in [1] to model triangulations with an unbounded face in the plane by inductive constructions. Then the Euler formula is proved by structural induction, before a nice proof of the five colour theorem is developed by induction of the graph size. The chosen model of triangulations is not completely combinatorial because the design of the constructors presupposes some planar properties related to the Jordan curve theorem, and distinguishes an external face which has to be specifically managed. This leads to some troubles and weakens the trust in the proof of the Euler formula. Anyway, it is done for triangulations only.

Hypermaps are precisely the combinatorial central structures used by Gonthier in [21] to prove in Coq the four colour theorem for a planar subdivision. In this work, each hypermap is described by a set equipped with three permutations, one of them being a composition of the other two. Hypermap constructors, which are inspired from a work by Stahl [43], are based on transpositions. The Euler formula is proved in an elementary case, by counting nodes, edges and faces on a rectangular grid, which discretizes the subdivision, then extended to the continuous plane

Table 1
Permutations α_0 and α_1 of the hypermap in Fig. 1

D	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
α_0	6	9	5	3	4	1	7	12	10	2	8	11	14	13	15
α_1	2	3	4	1	6	7	8	9	5	11	10	12	14	13	15

by classical topological arguments. The Euler formula is used as a global characterization of planarity, but a new local criterion, called *hypermap Jordan property* and easier to handle, is proved to be equivalent. Of course, the main part of this work is the titanic proof of the four-colour theorem following the pioneer proofs, but with hypermaps and sophisticated proof techniques.

This is an impressive result. However, another point of view on hypermaps could have been adopted. First, using three permutations creates a redundancy which makes specifications and proofs heavier. Second, the primitive constructors of S. Stahl are neither intuitive nor atomic. A greater simplicity would be welcome. Third, the proof of the Euler formula, first on a regular rectangular grid, next in aggregates of squares (called *mattes*), finally in the continuous regions of a map by passage to the limit, is rather disconcerting. Assuming as an *axiom* that combinatorial orientated maps or hypermaps exactly model polyhedra − planar subdivisions being a particular case − and directly proving properties of the genus could make the work simpler. That is what we will do in the following.

3. Mathematical aspects

3.1. Polyhedra and hypermaps

In this work, we adopt the following definition for the polyhedra.

Definition 1 (*Polyhedron*). A *polyhedron* is the subdivision of an orientatable closed surface into vertices (identified to points), edges (homeomorphic to open Jordan arcs) and faces (homeomorphic to open disks).

The topology of such a subdivision can be described by a hypermap or by a combinatorial orientated map.

Definition 2 (*Hypermap and Combinatorial Orientated Map*). (i) A *hypermap* is an algebraic structure $M = (D, \alpha_0, \alpha_1)$, where D is a finite set, the elements of which are called *darts*, and α_0, α_1 are permutations on D.

(ii) When α_0 is an involution without fixpoint on D, i.e. $\alpha_0(\alpha_0(x)) = x$ and $\alpha_0(x) \neq x$ for all x in D, then M is called a *combinatorial orientated map*.

(iii) If $y = \alpha_k(x)$, y is the *k-successor* of x, x is the *k-predecessor* of y, and x and y are said to be *k-linked*, or *k-sewn*, together.

Thus the combinatorial orientated maps, in short *maps*, form a subclass of the class of hypermaps.

Example 3 (*Hypermaps and Maps*). Let $D = \{1, \ldots, 15\}$. Table 1 gives α_0 and α_1, which are permutations, i.e. one-to-one correspondences, on D. Consequently, $M = (D, \alpha_0, \alpha_1)$ is a hypermap. In Fig. 1, M is drawn on the plane by associating to each dart an orientated arc of curve beginning with a bullet and ending with a small stroke. Darts which are 0-sewn (resp. 1-sewn) share the same small stroke (resp. bullet).

At this stage, exact shapes of curves or places of bullets and strokes in the plane are not important. We focus on topology, i.e. dart incidences and adjacencies symbolized by bullets and strokes. A convention we always adopt in drawings is that k-successors turn counterclockwise in the plane around strokes and bullets. Of course, in a combinatorial orientated map, there are exactly two darts incident to the same small stroke.

Our hypermap definition allows the void hypermap, i.e. with $D = \emptyset$, and fixpoints with respect to k, i.e. darts x such that $\alpha_k(x) = x$. In geometrical modelling, these particular cases can raise difficulties, especially fixpoints which generate *hanging edges*. They could be avoided, as in the map definition for the fixpoints, but they turn out to be relevant at early stages of specification and implementation. Let us give some examples, in which a notion of faces is intuitively used. A precise definition will be given immediately hereafter.

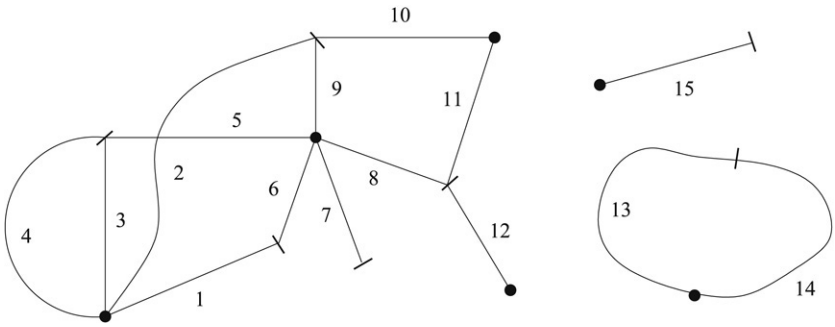
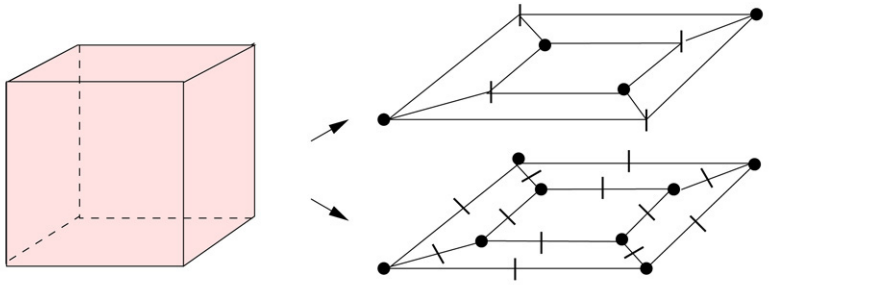


Fig. 1. An example of hypermap.



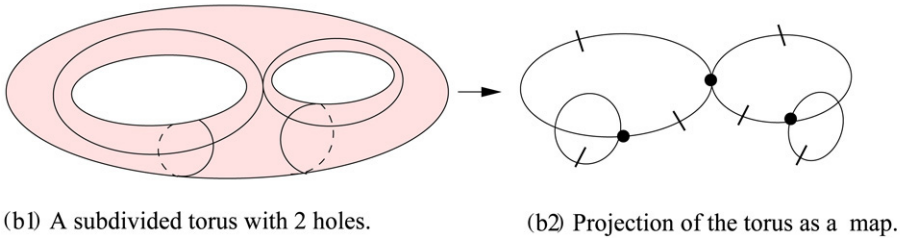


Fig. 2. Polyhedra and associated maps.

Example 4 (*Polyhedra and Maps*). Let us consider the polyhedra in Fig. 2. They can be described by hypermaps or maps drawn on the plane. To obtain them from a surface subdivision, we can perform a stereographic projection from a viewpoint near a face onto a plane on the other side of the subdivision. Thus, the cube is projected as a hypermap, where each edge is a dart, or as a map, where each edge is represented by two 0-sewn darts. Note that we consider that the polyhedral face close to the viewpoint is projected as an *external unbounded face*. When the polyhedron is disconnected, we must admit that the corresponding disconnected (hyper) map has one external face per connected component. This way, we have the right number of faces. The subdivision of the torus with two holes is projected as a map with intersection of dart representations, self-intersections being unavoidable in this case.

3.2. Cells of hypermaps

The topological cells of a hypermap − edges, vertices, faces and connected components −, which are also the cells of the underlying polyhedron, can be easily obtained from the permutations using a classical notion of orbit.

Definition 5 (*Orbits and Hypermap Cells*). (i) Let D be a set and Π be a set of functions in D. The *orbit* $\langle \Pi \rangle(x)$ of $x \in D$ for Π is the subset of D, the elements of which are accessible from x by any composition of functions of Π.

(ii) In hypermap $M = (D, \alpha_0, \alpha_1)$, $\langle \alpha_0 \rangle(x)$ is the *edge* of dart x, $\langle \alpha_1 \rangle(x)$ its *vertex*, $\langle \alpha_1^{-1} \circ \alpha_0^{-1} \rangle(x)$ its *face*, and $\langle \alpha_0, \alpha_1 \rangle(x)$ its *connected component*.

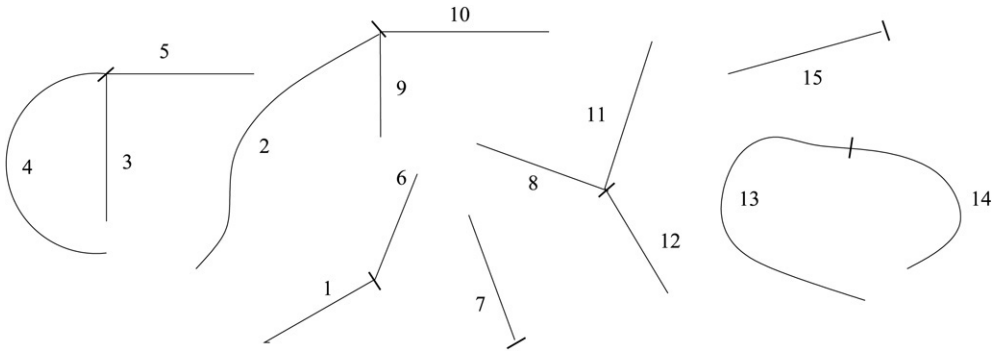


Fig. 3. Edges of the hypermap example.

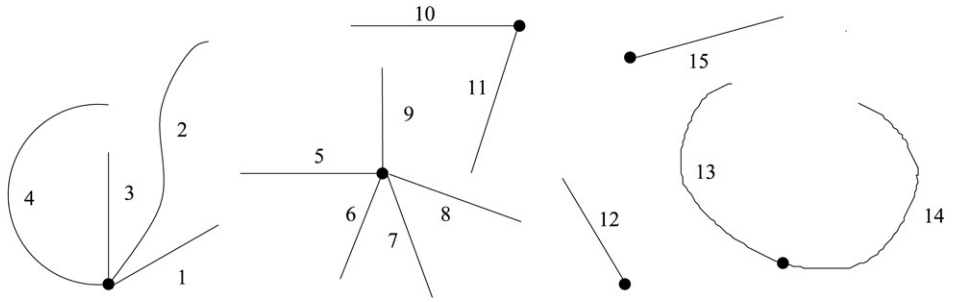


Fig. 4. Vertices of the hypermap example.

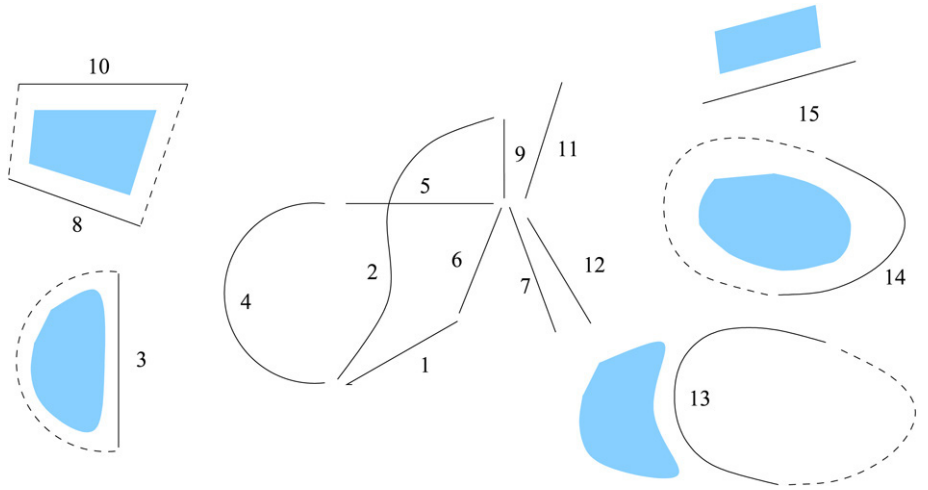


Fig. 5. Faces of the hypermap example.

Example 6 (*Hypermap Cells*). In Fig. 3 are drawn separately the 7 edges of the hypermap example from Fig. 1, in Fig. 4 its 6 vertices and in Fig. 5 its 6 faces. Dashed lines are added to symbolize which portion of the plane is concerned by the face. The face a dart belongs to corresponds to the portion marked by a grey spot on the left of the orientated dart representation. Thus, some face are bounded − or internal −, and others are unbounded − or external −. For instance, each of the cube projections of Fig. 2 contains 6 faces, 5 internal and 1 external. In Fig. 5, the face of dart 1 is rather surprising since it contains darts 1, 5, 2, 11, 12, 7, 6, 4, 9 and is neither internal nor external. This peculiarity is due to the fact that the hypermap is nonplanar (see below). The hypermap has of course 3 connected components.

Since α_0 and α_1 are permutations, it is clear that, for $\Pi = \{\alpha_0\}$, $\{\alpha_1\}$, $\{\alpha_1^{-1} \circ \alpha_0^{-1}\}$, or $\{\alpha_0, \alpha_1\}$, $y \in \langle\Pi\rangle(x)$ is equivalent to $x \in \langle\Pi\rangle(y)$. Faces are defined by $\alpha_1^{-1} \circ \alpha_0^{-1}$ for a dart traversal in counterclockwise order, similar to the other orbits, with our convention of plane representation.

3.3. Euler characteristic and planarity

Let M be a hypermap, and d, e, v, f, c be its numbers of darts, edges, vertices, faces, connected components, respectively.

Definition 7 (*Euler Characteristic, Genus, Planarity*). (i) The *Euler characteristic* of M is $\chi = v + e + f - d$.
(ii) The *genus* of M is $g = c - \chi/2$.
(iii) When $g = 0$, the hypermap is said to be *planar*.

This definition of χ is a bit more general than the one provided in the introduction because of the k-fixpoints of M. Of course, in a combinatorial orientated map with $D \neq \emptyset$, we have $d = 2 * e$ and $\chi = v - e + f$ as usual.

Example 8 (*Countings*). In Fig. 1, we have $d = 15$, $v = 6$ (number of bullets), $e = 7$ (number of strokes), $f = 6$, $c = 3$. Thus $\chi = 6 + 6 + 7 - 15 = 4$ and $g = c - \chi/2 = 1$. The hypermap in Fig. 1 is nonplanar. In Fig. 2, the map resulting from the projection of the torus subdivision which verifies $\chi = 3 + 6 + 1 - 12 = -2$ and $g = 1 - \chi/2 = 2$ is nonplanar. Note that this subdivision has one face only. Finally, the map and the hypermap coming from the cube subdivision are planar, with $g = 1 - (4 + 4 + 6 - 12)/2 = 0$ and $g = 1 - (8 + 12 + 6 - 24)/2 = 0$, respectively.

Note that, in all our examples, χ always is even, but may be negative, and that g is an integer which remains nonnegative. This is all the savour of the genus theorem.

Theorem 9 (*Of the Genus*). (i) *χ is an even integer.*
(ii) *g is a natural number.*

Of course, like Jacques [25] and Tutte [45], we have deliberately chosen a purely combinatorial genus definition. Here, the Euler formula can be considered as an alternative definition of planarity. For hypermaps, it is just a rewriting of the condition $g = 0$, and, for combinatorial orientated maps, it is written in its usual form with $c = 1$.

Definition 10 (*Euler Formula*). (i) A planar hypermap satisfies $\chi = v + e + f - d = 2 * c$.
(i) A nonempty planar-connected combinatorial orientated map satisfies $\chi = v - e + f = 2$.

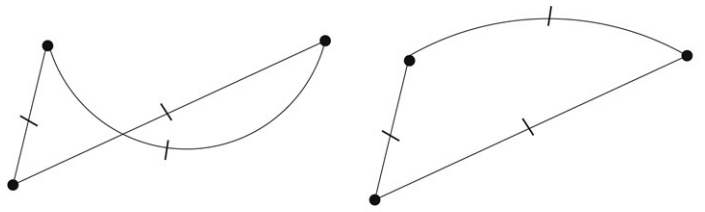
Example 11. (*Genus and Euler Formula*) In Fig. 2, both the map and the hypermap stemming from the cube subdivision satisfy the Euler formula. In Fig. 1, the hypermap is nonplanar and does not satisfy the Euler formula.

Under these conditions, what is a *proof* of the Euler formula? Here, it mainly consists to say which polyhedra − or (hyper) maps − satisfy the formula. Then, in the following, the focus is on the genus theorem and on the discovery of a condition of planarity for the hypermaps.

3.4. Embeddings

Let us consider hypermaps with nonempty dart sets D and α_0, α_1 without fixpoint. The *representation* of such a hypermap on an *orientatable closed* surface is a mapping of edges and vertices onto points, darts onto open oriented Jordan arcs, and faces onto regions homeomorphic to open disks. It is an *embedding* when each component of the hypermap generates a partition of the surface. As a result, an embedding must not contain any *self-intersection*, i.e. superposition of two vertex embeddings, intersection, partial or complete superposition of two edge embeddings, intersection of two face embeddings, etc.

For polyhedra, the genus g is classically interpreted as the number of *holes* in the underlying surface, or as the maximum number of surface cuts along Jordan curves without disconnection. It is a *topological invariant* of the surface and it can be computed from any subdivision of the surface. If $g = 0$, the polyhedra has the topology of the sphere. It can be projected onto a plane as a subdivision without self-intersection but with an *external unbounded face*. Otherwise $g > 0$, and the polyhedra has the topology of the torus with g holes. It cannot be projected on a plane without self-intersection.



(a) Non planar representation. (b) Planar representation (embedding).

Fig. 6. Representations of a planar map.

In the same way, each hypermap or map component can have in the Euclidean 3D space $\mathbb{R}^3$ an embedding on an orientatable closed surface, like a sphere or a torus with a sufficient number − the (hyper) map's genus g − of holes. When the (hyper) map is not connected, the surface is not connected either. Consequently, we have a natural subdivision of the surface, i.e. a polyhedron. Conversely, each polyhedral subdivision of an orientatable closed surface can be represented by a map or a hypermap, which entails the following result.

Fact 12 (*Correspondence Between Polyhedra and Hypermaps*). (i) *Any polyhedron can be represented by a hypermap whose dart set is nonempty and* α_0, α_1 *are without fixpoint.*

(ii) *Any such hypermap can be embedded in* $\mathbb{R}^3$ *as a polyhedron.*

Therefore, each definition or property set for polyhedra is transposable for (hyper) maps and vice versa. This is also the sense of the original note of Edmonds in [17]. This fact is often regarded as "well known" by mathematicians and not precisely proved [45]. In the formal specification which follows, it is never used. We can consider it as the *unique (informal) assumption* which connects our combinatorial results to usual polyhedra.

Moreover, we do not eliminate $D = \emptyset$ or fixpoints for α_0, α_1, which do not cause any trouble in our development. Indeed, when $D = \emptyset$, the hypermap is empty and the Euler formula holds with $\chi = 2 * c$. When there are fixpoints for α_0 or α_1, faces are embedded on connected open regions, and the previous numbering results still apply.

Example 13 (*Genus and Planarity*). The hypermap in Fig. 1 and the map resulting from the projection of the torus subdivision in Fig. 2 are nonplanar. They can never be drawn on the plane without self-intersections. Note that, although it has one face only, the torus subdivision is not *minimal*: one could find a subdivision with only one vertex and three incident loops, for instance two around the holes and one around the tube which separates them. Conversely, the map and hypermap stemming from the cube subdivision are planar and can be drawn on the plane without self-intersection. Since $g = 1$ for the non-planar hypermap in Fig. 1 (from where darts 7, 12 and 15 are removed), one may say that the underlying surface has the topology of the usual torus with one hole.

In the Euclidean plane $\mathbb{R}^2$, or in the sphere, only planar hypermaps and maps can have an embedding. But, without any precaution, even respecting our conventions of representation, they can also be represented with self-intersections.

Example 14 (*Planarity and Embedding*). Fig. 6 gives an example of a planar map which can be represented in the plane with or without a self-intersection.

Note that, for subdivisions of nonorientatable or open surfaces, the problem is more difficult. But they can always be embedded in a Euclidean 4D space. Prototype surfaces of these kinds are the Moebius band, which is an open non-orientatable surface embeddable in $\mathbb{R}^3$, and the Klein bottle, which is a closed nonorientatable surface only embeddable in $\mathbb{R}^4$.

Now, we consider the proof of the genus theorem and of Euler's formula for hypermaps. We have seen that the usual idea is to reason by induction on polyhedra cells — vertices, edges, or faces. Our creed is that the reasoning can be driven by structural induction on the hypermaps, provided that such objects can be constructively defined as an inductive type of terms. However, a gap exists between such an inductive definition and the true notion of hypermap. Adapting definitions and filling this gap is the main objective of the following section.

4. A formal map hierarchy

4.1. Preliminary specifications

Our map hierarchy requires some basic specifications which directly use notions that are built-in in Coq, as in any proof system. We first define a type `dim` for the two dimensions at stake, which we code `di0` and `di1`. In Gallina, the specification language of Coq, the declaration of this enumerated inductive type can be written

```
Inductive dim:Set:= di0: dim | di1: dim.
```

Since all objects are typed in Gallina, `dim` is declared with the type `Set` of all concrete types, and `di0`, `di1` with the type `dim`, as they are the *constructors* of this type. In Coq, the constructors of a type are considered as injective independent functions. Thus, `dim` is viewed exactly as the set formed by the distinct constant terms `di0` and `di1`.

At this stage, a first property can be proved, namely the *decidability* of the equality in `dim`. Recall that, the logic of Coq being intuitionistic, the *excluded middle* axiom is not built-in. As we want to remain in a strict intuitionistic framework, if it is necessary, the decidability of any predicate must be proved. The equality predicate = is built-in for each inductive type, but not its decidability. For `dim`, the latter can be established as a lemma.

```
Lemma eq_dim_dec:
  forall i j : dim, {i=j}+{~i=j}.
```

In Gallina, the decidability of = for dimensions, in the strong sense of distinguishing between `i=j` and `i<>j`, is conventionally written `{i=j}+{~i=j}`, which is a notation for a sum type itself of type `Set`. In accordance with the paradigm *proof as program* of intuitionistic type theory, when established, the proof is a function called `eq_dim_dec`, with 2 arguments, `i` and `j` of type `dim`, the result being of the sum type above. An object of this type can be tested in an `if ... then ... else ...` conditional expression.

This lemma is interactively proved with the help of some tactics implementing inference rules. For conciseness, they are not given in the paper. The reasoning is a structural induction on both `i` and `j`, here in fact a simple reasoning by case analysis. The latter justifies the definition of a specific type `dim` instead of considering dimensions as a subtype of the built-in naturals. Indeed, from each inductive type definition, Coq generates an *induction principle*, which can be used either to prove propositions or to build total functions on objects of this type.

Then, the type `dart` of darts is defined. The only constraints is that the set of darts is countable and has a decidable equality. So, it could be a Coq parameter type. For simplicity, we have chosen to identify `dart` with `nat`, the built-in inductive type of naturals. The decidability `eq_dart_dec` of the dart equality is a renaming of `eq_nat_dec` which is the built-in decidability of the equality in `nat`. To manage exceptions, a `nil` dart is defined as an alias of `0`.

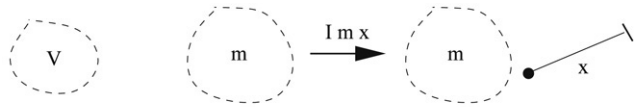
```
Definition dart:= nat.
Definition eq_dart_dec:= eq_nat_dec.
Definition nil:= 0.
```

Note that, in real implementations of geometrical map-based modellers, darts are often natural indices of arrays or pointers on structures, which justifies our choice.

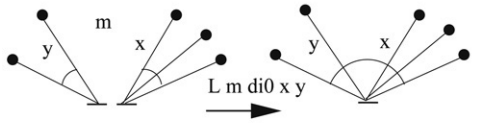
4.2. Free maps

Hypermaps are now approached by a notion of *free map*. Indeed, considering free algebras first is a general trick when dealing with inductive specification and reasoning. The definition of the type `fmap` of the free maps is written in Gallina as follows:

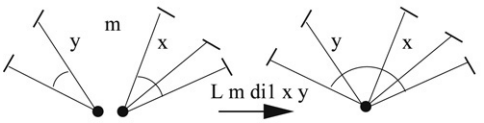
```
Inductive fmap:Set:=
      V : fmap
    | I : fmap->dart->fmap
    | L : fmap->dim->dart->dart->fmap.
```



(a) Empty map. (b) Insertion of an isolated dart.



(c) Linking of 2 darts at dimension 0.



(d) Linking of 2 darts at dimension 1.

Fig. 7. Action of the constructors.

Once again, it is an inductive type with 3 constructors, `V`, `I` and `L`, respectively for the empty (or void) hypermap, the insertion of a dart in a hypermap, and the linking of two darts in a hypermap. Their action is illustrated in Fig. 7, where sewings are represented by arcs of circle around strokes and bullets. In fact, the choice of these constructors dates back from the time when we formally developed our modeller using algebraic specifications [12,4,13]. It is crucial that we should keep the same basic framework to conduct our formal proofs in order to maintain an effective link with software construction.

In Coq, `fmap` is considered as the smallest set of ground terms which can be built from `V` applying `I` and `L`, considered as independent injections. Again, Coq generates an induction principle based on these constructors to prove propositions and build total functions on free maps. The direct application of these operations without restriction allows us to build rather complex objects, even more general than 2-graphs. Indeed, the same dart may be inserted several times by `I`. Even without insertion into the free map, a dart may be involved by `L` in several links at the same dimension `k`, with more than one `k`-successor or `k`-predecessor. Constraints will come later.

Example 15 (*A Hypermap Term*). A part of the hypermap in Fig. 1 is described by the term `m3`, where `m1, m2` are subterms:

```
m1 := I (I (I (I V 3) 4) 5) 2.
m2 := L (L (L m1 di0 4 3) di0 5 4) di1 2 3.
m3 := L (L (I m2 1) di1 1 2) di1 4 1.
```

The corresponding object is represented in Fig. 8. It is easy to understand how an entire hypermap can be built this way.

Our constructors are atomic, intuitive and independent from any embedding hypothesis. They presuppose absolutely no knowledge about the symmetrical group and work on two permutations only. They do not need to mention any particular cell as an external face. They can easily be combined to recover the other propositions of primitives in [26,44,45,33,31,43,46,1,21]. They offer a particularly simple induction to define operations and prove properties. Moreover, in our implementations, a (hyper-)map is always represented by a linear linked dart list, with pointers to other darts in order to represent the α_k and their inverses. Then, the constructors `V`, `I` and `L` are really programmed as elementary list manipulations. See for instance [4,15]. In exchange, our choice will force to progressively introduce some auxiliary notions and constraints, in order to catch and prove good properties of the permutations. This increases the size of the formal development.

Observers of free maps can be defined from this definition. The most immediate is the predicate `empty` which tests if a hypermap is empty or not. Its definition is made by case analysis thanks to a pattern matching on `m` written

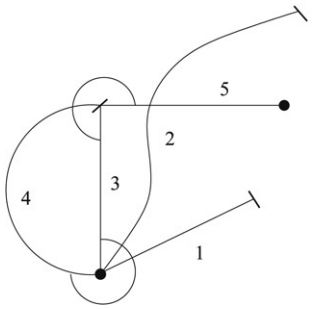


Fig. 8. Object corresponding to term m3.

`match m with ...`. The result is `True` or `False`, depending on the fact that `m` is `V` or anything else symbolized by `_`, the anonymous variable. `True` and `False` are the basic constants of `Prop`, the built-in type of propositions. A proof `empty_dec` of the decidability of `empty` by induction on `m` directly follows.

```
Definition empty(m:fmap): Prop:=
 match m with
       V => True
    |  _ => False
 end.

Lemma empty_dec: forall (m:fmap),
  {empty m}+{~empty m}.
```

In the same way, the predicate `exd` tests whether a dart exists in a hypermap, and its decidability `exd_dec` is derived by induction on `m`. This time, the definition is recursive, which is indicated by the keyword `Fixpoint`, by a pattern matching on `m`. At the end of the parameter list, `{struct m}` provides a hint to the proof system to check that the recursive calls are performed on smaller terms, which ensures termination. Note that terms are in prefix notation with as few parentheses as possible.

```
Fixpoint exd(m:fmap)(x:dart){struct m}:Prop:=
 match m with
       V => False
    |  I m0 x0 => x=x0 \/ exd m0 x
    |  L m0 _ _ _  => exd m0 x
 end.

Lemma exd_dec: forall (m:fmap)(x:dart),
  {exd m x}+{~exd m x}.
```

If, for each dart `x` and dimension `k`, we restrict our vision of a free map only to the last inserted `k`-links involving `x`, a free map can be viewed as an incomplete hypermap. Then, a partial version of operation α_k of the mathematical definition can be defined, but completed for convenience with `nil`. It is denoted `A` and its inverse `A_1`. In these definitions, note the use of the decidability functions `eq_dim_dec` and `eq_dart_dec` in conditional expressions, comments being written between `(*` and `*)`.

```
Fixpoint A(m:fmap)(k:dim)(x:dart){struct m}:dart:=
 match m with
      V => nil
    | I m0 x0 => A m0 k x
    | L m0 k0 x0 y0 =>
          if (eq_dim_dec k k0)
          then if (eq_dart_dec x x0) then y0
```

```
              else A m0 k x
         else A m0 k x
 end.

Fixpoint A_1(m:fmap)(k:dim)(x:dart){struct m}:dart:=
 (* Similar, but returns x0 instead of y0 *).
```

Auxiliary predicates `succ` and `pred` test whether a dart has a `k`-successor and a `k`-predecessor, with the corresponding lemmas, and functions, of decidability. The proofs are done by case analysis, e.g. for `succ_dec`, depending on the fact that `(A m k x)` is nil or not.

```
Definition succ(m:fmap)(k:dim)(x:dart):= A m k x <> nil.
Lemma succ_dec: forall (m:fmap)(k:dim)(x:dart),
  {succ m k x} + {~succ m k x}.
Definition pred(m:fmap)(k:dim)(x:dart):=
  A_1 m k x <> nil.
Lemma pred_dec: forall (m:fmap)(k:dim)(x:dart),
  {pred m k x} + {~pred m k x}.
```

Example 16 (*Use of Operations*). In object `m3` of the previous example, we have, with `<->` denoting the logical *equivalence*

```
A m3 di1 1 = 2; A m3 di1 2 = 3; A m3 di1 3 = nil.
A_1 m3 di0 5 = nil; A_1 m3 di0 3 = 4; A_1 m3 di0 4 = 5.
succ m3 di0 5 <-> True; succ m3 di1 3 <-> False.
pred m3 di1 4 <-> False; pred m3 di0 3 <-> True.
```

Finally, *destructors* are recursively defined. First, given a dart `x`, `D` deletes its latest insertion by `I`. Second, given a dart `x`, `B` and `B_1` break its latest `k`-link insertion by `L`, forward and backward respectively.

```
Fixpoint D(m:fmap)(x:dart){struct m}:fmap:=
  match m with
      V => V
    | I m0 x0 =>
        if (eq_dart_dec x x0) then m0
        else (I (D m0 x) x0)
    | L m0 k0 x0 y0 => (L (D m0 x) k0 x0 y0)
 end.

Fixpoint B(m:fmap)(k:dim)(x:dart){struct m}:fmap:=
  match m with
      V => V
    | I m0 x0 => (I (B m0 k x) x0)
    | L m0 k0 x0 y0 =>
        if (eq_dim_dec k k0)
        then if (eq_dart_dec x x0) then m0
             else (L (B m0 k x) k0 x0 y0)
        else (L (B m0 k x) k0 x0 y0)
  end.

Fixpoint B_1(m:fmap)(k:dim)(y:dart){struct m}:fmap:=
  (* Similar, but with (eq_dart_dec y y0) *).
```

Some lemmas can be quickly proved, expressing the effect of `D`, `B` and `B_1` on the previous predicates. For instance, we have the equivalence between the existence of darts in free maps `m` and `(B m k x)`.

```
Lemma exd_B: forall (m:fmap)(k:dim)(x y:dart),
   exd m y <-> exd (B m k x) y.
```

However, when they are applied without care, these operators can produce anything, including objects difficult to interpret and reuse. To obtain correct objects, i.e. satisfying some invariants, the operators have to be constrained using preconditions. We discuss this point in the next section.

4.3. Quasi-hypermaps

Free maps are objects with too many degrees of freedom to be safely managed in the subsequent proofs. Hence natural constraints are introduced on the constructors using preconditions. Each of them is presented as a predicate on the constructor's parameters. First, `prec_I m x` imposes that `x` inserted in `m` is different from `nil` and does not exist in `m`. Second, `prec_L m k x y` imposes that `x` and `y` both exist in `m`, `x` being without `k`-successor and `y` without `k`-predecessor.

```
Definition prec_I(m:fmap)(x:dart):=
  x <> nil /\ ~ exd m x.
Definition prec_L(m:fmap)(k:dim)(x y:dart) :=
  exd m x /\ exd m y /\ ~ succ m k x /\ ~ pred m k y.
```

If `I` and `L` are always used under these conditions, then the free map built is rather close to a hypermap, which can be considered as incomplete. This is the reason why it is called a *quasi-hypermap*. It satisfies an invariant `inv_qhmap` defined recursively

```
Fixpoint inv_qhmap(m:fmap){struct m}:Prop:=
  match m with
      V => True
    | I m0 x0 => inv_qhmap m0 /\ prec_I m0 x0
    | L m0 k0 x0 y0 => inv_qhmap m0 /\ prec_L m0 k0 x0 y0
  end.
```

Then, in Coq, it is possible to exactly define the type `qhmap` of the quasi-hypermaps using a familiar mathematical notation:

```
Definition qhmap:Set:= {m:fmap | inv_qhmap m}.
```

In type theory, and in Coq, a term of type `qhmap` is in fact a pair `(m, p)`, formed by a free map `m` and a proof `p` − depending on `m` − that `m` satisfies `inv_qhmap`. In the following, the `qhmap` type is not used, in order to avoid the destructuring of such a pair each time we have to deal with the underlying free map and invariant proof, which often occurs in inductive proofs.

Example 17 (*Quasi-Hypermap*). A quasi-hypermap is drawn in Fig. 9. Arcs of circles symbolize partial orbits and circles complete ones. It is sufficient to close each partial orbit, using only one `L`-link, to come back to the hypermap in Fig. 1. Note that `m3` in Fig. 8 was already a quasi-hypermap.

In `qhmap`, many properties about `A` and `A_1` can be proved. A first series concerns their relationships with the other observers. For instance, we have the following short lemmas:

```
Lemma not_exd_nil: forall m:fmap,
   inv_qhmap m -> ~ exd m nil.
Lemma succ_exd: forall (m:fmap)(k:dim)(x:dart),
   inv_qhmap m -> succ m k x -> exd m x.
Lemma A_nil: forall (m:fmap)(k:dim),
   inv_qhmap m -> A m k nil = nil.
```

A second series concerns the fact that, in a quasi-hypermap, `A` and `A_1` are inverses, which was generally false in free maps.

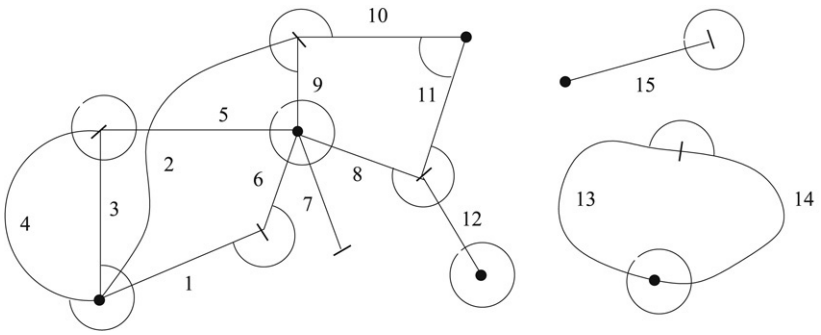


Fig. 9. Quasi-hypermap for the example hypermap.

```
Lemma A_A_1: forall (m:fmap)(k:dim)(x y:dart),
   inv_qhmap m -> exd m x -> exd m y
     -> y = A m k x -> x = A_1 m k y.
Lemma A_1_A: forall (m:fmap)(k:dim)(x y:dart),
   inv_qhmap m -> exd m x -> exd m y
     -> x = A_1 m k y -> y = A m k x.
```

Finally, we can prove that `A` and `A_1` are *partial injections*. More precisely, in Coq, we can generically define, for a partial function `f:dart->dart` the notion of "*being an injection on a definition domain*", the domain being given by a unary relation `Df:dart->Prop`.

```
Definition inj_dart(Df:dart->Prop)(f:dart->dart):Prop:=
   forall x x':dart,
       Df x -> Df x'-> f x = f x'-> x=x'.
```

Then we can prove that, for each quasi-hypermap `m` and dimension `k`, `(A m k)` is an injection on the domain defined by `(succ m k)`.

```
Lemma inj_A_qhmap :
   forall (m:fmap)(k:dim), inv_qhmap m ->
      inj_dart (succ m k) (A m k).
```

However, in quasi-hypermaps, `k`-orbits may be *incomplete*, in the sense that the series of iterated `k`-successors of a dart `x` can be interrupted by `nil`. Of course, we have the same properties for `A_1`.

4.4. Hypermaps

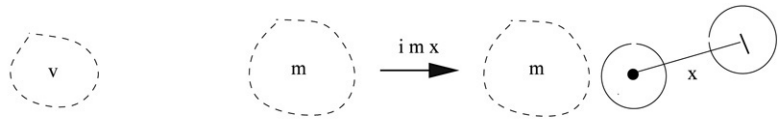
From a constructive point of view, a real hypermap can be considered as a *complete* quasi-hypermap, i.e. it is equipped with complete sewings. No `k`-successor, or `k`-predecessor, can be `nil`, or, which is equivalent, each dart of the hypermap has a real `k`-successor and a real `k`-predecessor. This is expressed by an invariant `inv_hmap` for the type `hmap` of the hypermaps.

```
Definition inv_hmap(m:fmap):Prop:= inv_qhmap m /\
  forall (x:dart)(k:dim), exd m x -> succ m k x /\ pred m k x.
```

The type `hmap` is a subtype of `qhmap`, but it turns out that considering it as a subtype of `fmap` is more tractable.

```
Definition hmap:Set:= {m:fmap | inv_hmap m}.
```

Then, it is possible to carry out the proof of the hypermap properties, namely the fact that α_k is a permutation, for $k = 0, 1$. We already know from Section 4.3 that `(A m k)` is an injection on domain `(succ m k)`. We prove now that this holds on domain `(exd m)`, and we have the same property for `A_1`.



(a) Empty hypermap. (b) Insertion of a new dart sewn to itself.

Fig. 10. Empty hypermap and insertion of a sewn dart.

```
Lemma inj_A: forall (m:fmap)(k:dim),
   inv_hmap m -> inj_dart (exd m) (A m k).
Lemma inj_A_1: forall (m:fmap)(k:dim),
   inv_hmap m -> inj_dart (exd m) (A_1 m k).
```

We also have to deal with surjectivity. Of course, mathematically speaking, it is well known that an injection on a finite set is also a surjection. In fact, from the hypermap specification, we can immediately prove that `(A m k)` is a surjection on `(exd m)`. We now specify the notion of *surjection*, and also of *permutation* on a dart domain.

```
Definition surj_dart(Df:dart->Prop)(f:dart->dart):Prop:=
  forall y:dart, Df y ->
     exists x:dart, Df x /\ f x = y.
Definition perm_dart(Df:dart->Prop)(f:dart->dart):Prop:=
  inj_dart Df f /\ surj_dart Df f.
```

We get the expected results for `(A m k)`, first the surjection, then the permutation. The same is obtained for `(A_1 m k)`.

```
Lemma surj_A: forall (m:fmap)(k:dim),
  inv_hmap m -> surj_dart (exd m) (A m k).
Theorem perm_A: forall (m:fmap)(k:dim),
  inv_hmap m -> perm_dart (exd m) (A m k).
```

Here, operations to directly build hypermaps, i.e. leading from hypermap to hypermap, can be defined using the constructors of free maps. First, operation `V` returns a free map, but also a hypermap. However, to be homogeneous with what follows `V` is hidden by `v`.

```
Definition v := V.
```

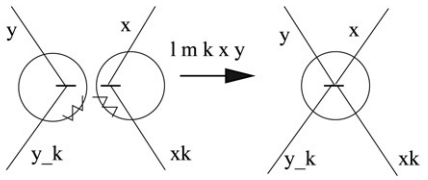
Second, operation `i` represents the insertion of a new isolated dart which is 0- and 1-sewn to itself, provided that precondition `prec_i`, in fact `prec_I`, is respected.

```
Definition prec_i(m:fmap)(x:dart):=
  prec_I m x.
Definition i(m:fmap)(x:dart):=
  L (L (I m x) di0 x x) di1 x x.
```

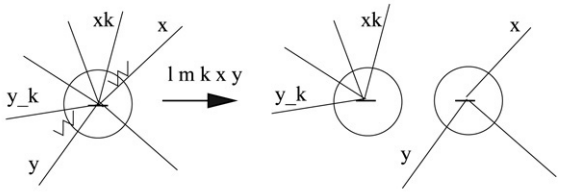
Example 18 (*Empty Hypermap and Insertion of a Dart in a Hypermap*). Fig. 10 illustrates the construction by `v` of the empty hypermap and the insertion by `i` of a new dart sewn to itself.

Third, operation `l` `k`-links dart `x` to dart `y` after their corresponding links have been broken. Moreover, in order to close their `k`-orbits, the `k`-predecessor of `y` is linked to the `k`-successor of `y`. This operation is useful, and possible, only if `y` is not already the `k`-successor of `x`.

```
Definition prec_l(m:fmap)(k:dim)(x y:dart) :=
   exd m x /\ exd m y /\ y <> A m k x.
Definition l(m:fmap)(k:dim)(x y:dart) :=
   let xk  := (A m k x) in
   let y_k := (A_1 m k y) in
```



(a) Merging two k-orbits into one k-orbit.



(b) Splitting one k-orbit into two k-orbits.

Fig. 11. Merge and split of edges.

```
let m1 := (B_1 (B m k x) k y) in
(L (L m1 k x y) k y_k xk).
```

In fact, `l` can merge two distinct `k`-orbits or split a `k`-orbit, depending on the fact that `x` and `y` are in different `k`-orbits of `m` or not.

Example 19 (*Linking Two Darts in a Hypermap: Merging vs Splitting*). Fig. 11 illustrates both cases of the use of `l m k x y` to sew darts `x` and `y` at dimension `k` in a hypermap `m`. The break of a sewing is symbolized by a small zigzag. The figure deals with edges, but it would be exactly the same for vertices.

In fact, it is easy to see that realizing a *transposition*, roughly speaking an exchange of two darts in a permutation, exactly needs two applications of `l`. Operations to deal with hypermaps in [25,26,44,45,33,43,21] are essentially based on transpositions. Simple as it may seem, a transposition is in fact a composed operation, which has to be broken down, in proving as well as in programming. Thanks to our `L` and `B` atomic operations, we have chosen to break it not only in proofs and programs, but already at the specification level. This allows to master all the small steps of calculus, and to make proofs and programs on hypermaps clearer and shorter. Conversely, always staying at the (real) hypermap level sometimes forces us to deal with long sequences of operations. At this stage, it is necessary to prove that `i` and `l`, when respecting their preconditions, preserve the invariant of the hypermaps. This is what is conventionally called *proof obligations*, the one for `l` being rather long (two hundred tactic applications).

```
Theorem inv_hmap_i: forall (m:fmap)(x:dart),
  inv_hmap m -> prec_i m x -> inv_hmap (i m x).
Theorem inv_hmap_l:
  forall (m:fmap)(k:dim)(x y:dart),
    inv_hmap m -> prec_l m k x y -> inv_hmap (l m k x y).
```

At this point, a type `map` for the *combinatorial orientated maps* is easy to define as a restriction of `hypermap`. An invariant `inv_map` can be written, the proof that `(A m di0)` is an *involution* for `map` can be made, and the precondition on `l` can be enforced to only work with `map`. However, we do not use combinatorial orientated maps in the following subsection.

4.5. Closure of a quasi-hypermap

Since a quasi-hypermap is a hypermap except for some incomplete `k`-orbits, the question at stake is the completion of a quasi-hypermap to yield to a hypermap. That is the role of the operation `clos`. Although intended for quasi-hypermaps, it is easily defined on free maps by a structural induction.

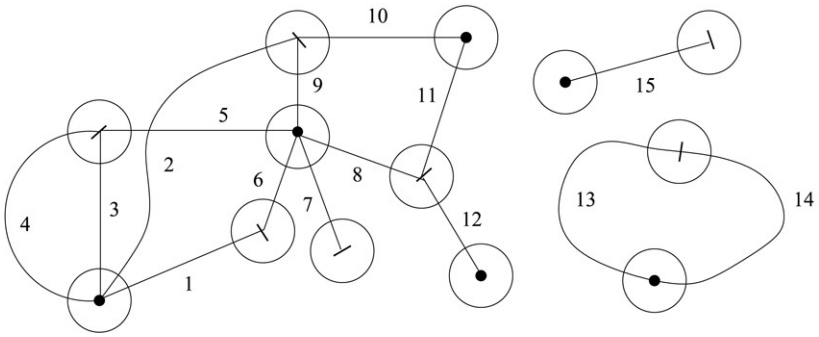


Fig. 12. Closure of the quasi-hypermap in Fig. 9.

```
Fixpoint clos(m:fmap):fmap:=
  match m with
    V => V
  | I m0 x0 =>
      i (clos m0) x0
  | L m0 k x0 y0 =>
      let mr:= clos m0 in
        if eq_dart_dec y0 (A mr k x0) then mr
        else l mr k x0 y0
  end.
```

Example 20 (*Quasi-Hypermap Closure*). If `m` is the quasi-hypermap in Fig. 9, `clos m` is the hypermap in Fig. 12 which exactly corresponds to Fig. 1.

Now, the expected properties of the closure have to be proved. One of the most important is that it transforms a quasi-hypermap into a real hypermap.

```
Theorem inv_hmap_clos: forall (m:fmap),
  inv_qhmap m -> inv_hmap (clos m).
```

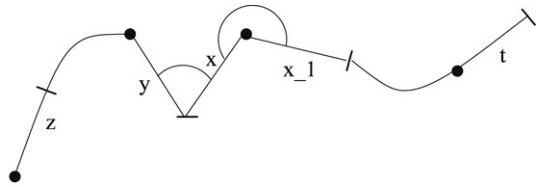
In addition, it has been shown that `clos` preserves the dart set of the quasi-hypermap and that it does exactly the closure of `k`-orbits: pre-existing `k`-successors are the same, and, when the orbit is incomplete, the `k`-successor of the `k`-orbit *head* is the `k`-orbit *tail*. The same process applies for `A_1`.

5. Paths and countings

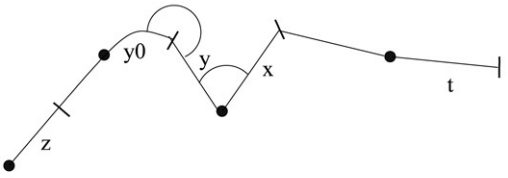
5.1. Paths in faces

Testing whether it is possible to go from a dart to another in a face is crucial to determine the number of faces. The operation `F` gives the successor of a dart in a face. It is defined as the composition of `A_1` at dimensions 0 and 1, exactly like in the mathematical definition. We also have defined the existence `succf` of a successor by `F` different from `nil`. It is easily proved to be decidable.

```
Definition F(m:fmap)(x:dart):=
  A_1 m di1 (A_1 m di0 x).
Definition succf(m:fmap)(x:dart):Prop:=
  pred m di0 x /\ pred m di1 (A_1 m di0 x).
Lemma succf_dec : forall (m:fmap)(x:dart),
    {succf m x}+{~succf m x}.
```



(a) Case L m di0 x y.



(b) Case L m di1 x y.

Fig. 13. Existence of a path from z to t.

The inverse, `F_1`, with `predf`, is defined similarly, and `predf_dec` proved as well. In fact, we have exactly the same properties as for `A` and `A_1`, for instance the proof that `F` and `F_1` are *permutations* and inverses of each other in hypermaps. However, we must be more careful with the premises, e.g.

```
Lemma F_F_1 : forall (m:fmap)(x:dart),
  inv_qhmap m -> exd m x -> exd m (F_1 m x) ->
     F m (F_1 m x) = x.
Lemma F_1_F : forall (m:fmap)(x:dart),
  inv_qhmap m -> exd m x -> exd m (F_1 m x) ->
     F_1 m (F m x) = x.
```

Then, the *existence* `expf` *of a path* following `F` from a dart to another in a hypermap face is inductively defined on free maps and its decidability proved:

```
Fixpoint expf(m:fmap)(z t:dart){struct m}:Prop:=
 match m with
     V => False
  |  I m0 x => expf m0 z t \/ z=x /\ t=x
  |  L m0 di0 x y =>
       expf m0 z t
     \/ let x_1 := A_1 m0 di1 x in
        x_1<>nil /\ expf m0 z y /\ expf m0 x_1 t
  |  L m0 di1 x y =>
       expf m0 z t
     \/ let y0 := A m0 di0 y in
        y0<>nil /\ expf m0 z y0 /\ expf m0 x t
 end.

Lemma expf_dec : forall(m:fmap)(x y:dart),
  {expf m x y} + {~expf m x y}.
```

Some explanations about `expf` definition, for patterns `L m0 di0 x y` and `L m0 di1 x y` are given in Fig. 13. Then, some properties of `expf` are proved. The *reflexivity* `refl_expf` and the *transitivity* `trans_expf` are rather easy to obtain by induction on free maps.

```
Lemma refl_expf : forall(m:fmap)(x:dart),
  exd m x -> expf m x x.
```

```
Lemma trans_expf : forall(m:fmap)(x y z:dart),
  expf m x y -> expf m y z -> expf m x z.
```

Conversely, the *symmetry* of `expf` is only satisfied with real hypermaps. In our present framework, the proof of this property is feasible and has been done partially, but its achievement is too complex in the general case. Fortunately, it is not used in the following proofs. Finally, some other properties of *dead end* and *subtraction* for paths have turned out to be particularly useful. Let us give two samples, which are not really immediate to obtain.

```
Lemma not_pred0_not_expf : forall(m:fmap)(x y:dart),
  inv_qhmap m -> ~pred m di0 x -> x<>y -> ~expf m x y.
Lemma diff1_expf : forall(m:fmap)(x y z:dart),
  inv_qhmap m -> expf m x y -> expf m x z ->
    (expf m y z \/ expf m z y).
```

It is necessary to express the *connectivity* of a quasi-hypermap, or of a free map, which is actually the same. A binary relation `eqc` stating that two darts belong to the same hypermap connected component is easily defined by induction, in the way of the famous *Warshall algorithm* dealing with the existence of paths in a directed graph. The difference is that this algorithm uses an induction on graph vertices while the following definition uses a structural induction on the hypermap structure. We also have the decidability of `eqc`.

```
Fixpoint eqc(m:fmap)(z t:dart){struct m}:Prop:=
 match m with
     V => False
  |  I m0 x => z=x /\ t=x \/ eqc m0 z t
  |  L m0 _ x y =>
      eqc m0 z t
     \/ (eqc m0 z x /\ eqc m0 y t)
     \/ (eqc m0 z y /\ eqc m0 x t)
 end.

Lemma eqc_dec: forall (m:fmap)(x y:dart),
   {eqc m x y} + {~eqc m x y}.
```

Using induction, we quickly obtain the proofs of *reflexivity*, *symmetry* and *transitivity* of `(eqc m)` for any hypermap `m`. A few useful lemmas express simple relationships between operations `A`, `A_1` and connectivity. For instance,

```
Lemma succ_eqc_A_r: forall m:fmap,
 inv_qhmap m ->
   forall (k:dim)(x:dart),
     succ m k x -> eqc m x (A m k x).
Lemma eqc_eqc_A_1_l: forall m:fmap,
 inv_qhmap m ->
   forall (k:dim)(x y:dart),
     eqc m x y -> pred m k x -> eqc m (A_1 m k x) y.
```

Finally, we have very useful variations around the fact that `eqc` is a consequence of `expf`

```
Lemma expf_eqc: forall m:fmap,
 inv_qhmap m ->
   forall (x y:dart), expf m x y -> eqc m x y.
Lemma expf_A_r_eqc: forall(m:fmap)(k:dim),
 inv_qhmap m ->
   forall (x y:dart), expf m x (A m k y) -> eqc m x y.
```

5.2. Characteristics

We are now ready to count the cells and components of a hypermap. As usual, the operations are designed for quasi-hypermaps where they make sense, but for convenience they are defined by induction on free maps. We work with a Coq library module named ZArith, which contains all the features for Z, the ring of the integers. The number nd of darts in a quasi-hypermap can immediately be defined by induction. It increases by 1 for pattern I m0 x only.

```
Fixpoint nd(m:fmap):Z:=
 match m with
    V => 0
  | I m0 x => nd m0 + 1
  | L m0 _ _ _ => nd m0
 end.
```

The definition of the number ne of edges is more difficult to read. Let us focus on the most interesting pattern L m0 di0 x y. If, in the closure (clos m0), y is already the 0-successor of x, then L m0 di0 x y only helps to close the 0-orbit common to x and y. Thus, the number of 0-orbits in the quasi-hypermap does not change. Otherwise, x and y are in different 0-orbits which are merged, and the number of 0-orbits decreases of 1. The case of the number nv of vertices is similar, replacing di0 by di1.

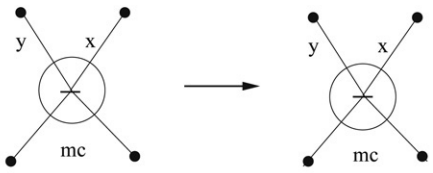
```
Fixpoint ne(m:fmap):Z:=
 match m with
    V => 0
  | I m0 x => ne m0 + 1
  | L m0 di0 x y => ne m0 -
      if eq_dart_dec (A (clos m0) di0 x) y
      then 0 else 1
  | L m0 di1 x y => ne m0
 end.

Fixpoint nv(m:fmap):Z:=
 match m with
    V => 0
  | I m0 x => nv m0 + 1
  | L m0 di0 x y => nv m0
  | L m0 di1 x y => nv m0 -
      if eq_dart_dec (A (clos m0) di1 x) y
      then 0 else 1
 end.
```

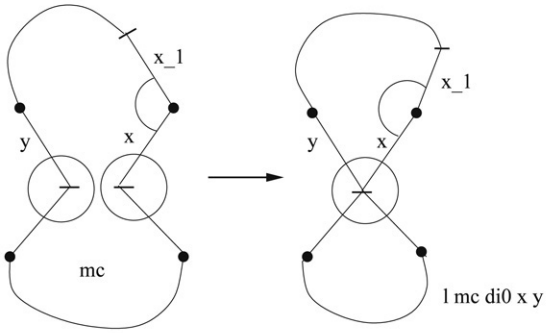
The definition of the number nf of faces is more complicated, except for the patterns V and I m0 x0, where it is trivial. Hence, let us consider the pattern L m0 di0 x y. Let mc, x0 and x_1 be defined as in the following specification by mc := clos m0, x0 := A mc di0 x, and x_1 := A_1 mc di1 x. There are three cases which are illustrated in Fig. 14:

- *Case* 1: y = x0. Then, the purpose of L m0 di0 x y is to close the 0-orbit common to x and y. Nothing arises in mc and with the number of faces.
- *Case* 2: y <> x0 and expf mc x_1 y, i.e. there is a path in a face of the closure mc from x_1 to y, and, by symmetry of expf in the closures, there is also a face path from y to x_1 in mc. Then, a new face is created, because the face common to x_1 and y in mc is split. Hence nf increases by 1.
- *Case* 3: y <> x0 and ~ expf mc x_1 y, i.e. x_1 and y are in different faces of mc. Then, both faces are merged, and nf decreases by 1.

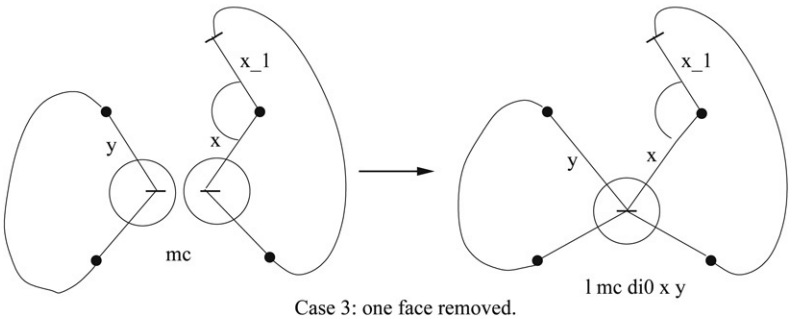
The case of pattern L m0 di1 x y is similar, with conditions y = x1 and expf mc x y0, and, in Fig. 14, x_1 and y replaced by x and y0.



Case 1: same faces.



Case 2: one more face.



Case 3: one face removed.

Fig. 14. Variation in the number of faces.

```
Fixpoint nf(m:fmap):Z:=
 match m with
    V => 0
  | I m0 x => nf m0 + 1
  | L m0 di0 x y =>
      let mc := clos m0 in
      let x0 := A mc di0 x in
      let x_1:= A_1 mc di1 x in nf m0 +
       if eq_dart_dec y x0 then 0
       else if expf_dec mc x_1 y then 1 else -1
  | L m0 di1 x y =>
      let mc := clos m0 in
      let x1 := A mc di1 x in
      let y0 := A mc di0 y in nf m0 +
       if eq_dart_dec y x1 then 0
       else if expf_dec mc x y0 then 1 else -1
 end.
```

Finally, the definition of the number `nc` of connected components does not pose any problem with the condition `eqc (clos m0) x y`. The definition in `Z` of the Euler characteristic immediately follows:

```
Fixpoint nc(m:fmap):Z:=
```

```
 match m with
    V => 0
  | I m0 x => nc m0 + 1
  | L m0 _ x y => nc m0 -
       if eqc_dec (clos m0) x y then 0 else 1
 end.

Definition ec(m:fmap):Z:=
  nv m + ne m + nf m - nd m.
```

5.3. Genus theorem and Euler formula

Everything is set to prove the first part of the genus theorem, which is the parity of `(ec m)` for a quasi-hypermap `m`, as well as for a free map `m`. Indeed, it is rather surprising that the proof does not require the invariant `(inv_qhmap m)`. During the proof, numerous small linear equations in `Z` must be verified. This is made automatically calling a Coq tactic named `omega`, which solves systems of equations, disequations and inequations in Presburger's arithmetic [37]. Useful properties of the parity predicate `Zeven` in `Z` are already present [7]. The second part of the genus theorem immediately follows, this time necessarily under the `(inv_qhmap m)` condition.

```
Theorem genus_1: forall m:fmap, Zeven (ec m).
Theorem genus_2: forall m:fmap,
  inv_qhmap m -> 2 * (nc m) >= (ec m).
```

The structure of the proofs of both subtheorems is the same. It is founded on an elementary structural induction on `m` and the linking dimension. The proofs can be outlined together as follows with four cases — recall that `ec m = nv m + ne m + nf m - nd m` for all fmap `m` —:

- *Case* 1: `m = V`. Since `nv V = ne V = nf V = nd V = nc m = 0` by definition, `ec m` is null, then even and such that `2 * (nc m) >= (ec m)`.
- *Case* 2: `m = I m0 x`, with the *induction hypothesis*: `even (ec m0)` and `2 * (nc m0) >= (ec m0)`. Then, from the previous specifications, `nv m = (nv m0) + 1`, and the same for `ne`, `nf`, `nd` and `nc`. Hence, `ec m = (ec m0) + 2`, which remains even, and we always have `2 * (nc m) >= (ec m)`.
- *Case* 3: `m = L m0 di0 x y`, with the *induction hypothesis*: `even (ec m0)` and `2 * (nc m0) >= (ec m0)`. Let `mc := clos m0` be the closure of `m0`. Two subcases arise (see Fig. 14 again):
  - *Case* 3.1: `(A mc di0 x) = y`, i.e. `x` has `y` as 0-successor in `mc`. Then, from the previous definitions, each characteristic keeps its value and the result follows.
  - *Case* 3.2: `~ (A mc di0 x) = y`, i.e. `x` has not `y` as 0-successor in `mc`. Then, `ne m = (ne m0) - 1` by definition, and, setting `x_1 := (A_1 mc di1 x)`, there are two subcases again.
    - *Case* 3.2.1: `expf mc x_1 y`, i.e. `x_1` and `y` are in the same face of `mc`. Then, from the previous definitions, the only variations of the characteristics are `ne m = (ne m0) - 1` and `nf m = (nf m0) + 1`. Hence `ec m = ec m0` and the result follows.
    - *Case* 3.2.2: `~ expf mc x_1 y`, i.e. `x_1` and `y` are not in the same face of `mc`. Then `nf m = (nf m0) - 1` by definition. Then, since `ne m = (ne m0) - 1`, we have `ec m = (ec m0) - 2`, which may be negative but remains even. At last, two subcases arise for `nc`.
      - *Case* 3.2.2.1: `eqc mc x y`, i.e. `x` and `y` are in the same connected component of `mc`. Then, `nc m = nc m0`, and the result follows.
      - *Case* 3.2.2.2: `~ eqc mc x y`, i.e. `x` and `y` are not connected in `mc`. Then, `nc m = (nc m0) - 1`, `ec m = (ec m0) - 2`, hence `2 * (nc m) >= (ec m)`.
- *Case* 4: `m = L m0 di1 x y`. The reasoning is exactly the same as in *Case* 3, replacing `di0`, `x0`, `x_1`, `y` by `di1`, `x1 := (A mc di1 x)`, `x`, `y0 := (A mc di0 y)`, respectively.

Next, the `genus` function can be defined and a corollary of `genus_2` is given

```
Definition genus(m:fmap):= (nc m) - (ec m)/2.
Theorem genus_corollary : forall m:fmap,
  inv_qhmap m -> genus m >= 0.
```

Finally, the planarity predicate `planar` is defined and the Euler formula is trivially obtained in its general form, by a specialization of the genus

```
Definition planar(m:fmap):= genus m = 0
Lemma Euler_formula: forall m:fmap,
  inv_qhmap m -> planar m -> ec m / 2 = nc m.
```

Now, it is crucial to be sure that a quasi-hypermap is planar and satisfies the Euler formula.

5.4. Sufficient conditions of planarity, proof of Euler's formula

From the above results, we can quickly prove that the empty free map is planar and infer sufficient conditions of planarity preservation when we − correctly − insert a dart, 0-sews or 1-sews two darts in a quasi-hypermap.

```
Lemma planar_V: planar V.
Lemma planar_I: forall (m:fmap)(x:dart),
  inv_qhmap m -> planar m -> prec_I m x ->
       planar (I m x).
Lemma planar_L0: forall (m:fmap)(x y:dart),
  inv_qhmap m -> planar m ->
    prec_Lq m di0 x y -> let mc:= clos m in
     (~eqc mc x y \/ expf mc (A_1 mc di1 x) y) ->
        planar (L m di0 x y).
Lemma planar_L1: forall (m:fmap)(x y:dart),
  inv_qhmap m -> planar m ->
    prec_Lq m di1 x y -> let mc:= clos m in
     (~eqc mc x y \/ expf mc x (A mc di0 y)) ->
        planar (L m di1 x y).
```

The last two precisely define a *sewing between two components* and a *sewing within a face* which preserve planarity. They allow us to express exactly what is *adding an edge within a face*, which we were aiming at in our introduction. It would be interesting to establish a link between this constructive conditions and the static one of [21]. Consequently, if we exclusively use the three constructors under these conditions, we build a planar quasi-hypermap. Beside `inv_qhmap`, such a quasi-hypermap satisfies `plf` − for *planarly-formed* −, a predicate recursively defined by

```
Fixpoint plf(m:fmap):Prop:=
  match m with
     V => True
   | I m0 x => plf m0
   | L m0 di0 x y => plf m0 /\
      (let mc := (clos m0) in
        ~eqc mc x y \/ expf mc (A_1 mc di1 x) y)
   | L m0 di1 x y => plf m0 /\
      (let mc := (clos m0) in
        ~eqc mc x y \/ expf mc x (A mc di0 y))
  end.
```

Then, it is easy to prove by induction on `m` that the planar formation ensures the planarity. We have thus a *constructive sufficient condition of planarity* which provides a *proof of the Euler formula*.

```
Theorem plf_planar:forall (m:fmap),
  inv_qhmap m -> plf m -> planar m.
Theorem plf_Euler_formula: forall m:fmap,
  inv_qhmap m -> plf m -> ec m / 2 = nc m.
```

5.5. Necessary conditions and characterization of planarity

Actually, `plf` is also a necessary condition of planarity, but the proof seems rather difficult to obtain in the present state. Thus, in this paper, we only *conjecture* the *reciprocal* of `plf_Euler_formula`.

```
Conjecture Euler_formula_plf: forall m:fmap,
  inv_qhmap m -> ec m / 2 = nc m -> plf m.
```

The achievement of this result is one of our next objectives. Finally, we will have proved that the subclass of the quasi-hypermaps which exactly model the planar polyhedra − or satisfy the *Euler formula* − is entirely characterized by the *constructive* predicate `plf`.

```
Conjecture Euler_formula_criterion: forall m:fmap,
  inv_qhmap m -> (plf m <-> ec m / 2 = nc m).
```

6. Conclusion

We have shed a new light on the proofs of the genus theorem and the Euler formula for polyhedra via the hypermap notion. They rely on simple arguments relating to properties of permutations in finite sets, which are proved *from scratch*. Indeed, there are *no axiom* at all, apart from Fact 12, which is external, and the *minimization* induced by the inductive definitions of `dim` and `fmap`. Such proofs involve a substantial framework of specification where a term algebra − the algebra of free maps −, plays a preeminent role, since it leads to a very simple induction. In this process, the inductive building and updating of permutations is very important. In addition, the atomic operations we have defined to achieve this aim allow to easily build by composition any map or polyhedra operation of the literature. The type of free maps must be constrained in order to precisely represent the objects we have in mind, first the quasi-hypermaps and then the hypermaps, these subtypes being defined thanks to invariants. At the same time, operations must be restricted using preconditions. In fact, this framework is basically the same as the one we have built to specify and develop geometrical modellers via algebraic specifications.

The Calculus of Inductive Constructions was appropriate to specify our hypermaps in a constructive way. Nevertheless, in this well delimited work, we have not used all its powerful features. The only drawback we have noticed concerns definitions by induction, which make it necessary to work syntactically with free term algebras. Instead, term algebras defined *modulo equations*, such as those available in algebraic specifications, would give more flexibility. The Coq system turned out to be a precious auxiliary to guide and check all the proofs. The development we have presented represents about 4500 lines of Gallina, including about 40 definitions, and 120 lemmas and theorems. But, once the patient construction of the map hierarchy was done, all the countings, including the proofs of the genus theorem and the Euler formula with the planarity conditions were brief − about 500 lines − and easy.

Future work will deepen the present topological results, particularly on planarity characterization for polyhedra, and, more generally, on homology for combinatorial varieties. Furthermore, we have begun to revisit the foundations of computational geometry using formal specifications, proofs, maps and hypermaps in the way of [14]. In this field, many notions are sheerly topological and derive from the methods we have presented. The others, which concern geometrical embedding, real numbers and round-off errors, are more difficult to tackle. We think that progress will come from appropriate axiomatizations of the numbers, or axiomatizations allowing us to bypass them, like the one by D. Knuth in [28] for orientation in the plane and convex hull, or by P. Schorn [42] for plane-sweep algorithms. Finally, we also intend to use another promising feature of provers which are based on Curry–Howard isomorphism, namely the extraction of programs from proofs. Such programs are automatically correct, or *certified*, with respect to formal specifications [30]. This way, we hope that we will be able to extract correct programs of computational topology or geometry from constructive proofs.

Acknowledgements

I would like to thank Franck Pfenning, Donald Sannella and the anonymous reviewers for the many remarks and suggestions that have helped to substantially improve this paper.

References

[1] G. Bauer, T. Nipkow, The 5 Colour Theorem in Isabelle/Isar, in: Theorem Proving in HOL Conf., in: LNCS, vol. 2410, Springer-Verlag, 2002, pp. 67–82.
[2] B. Baumgart, A boundary representation for computer vision, in: 44th AFIPS Nat. Conf., 1975, pp. 589–596.
[3] Y. Bertot, P. Castéran, Interactive theorem proving and program development — Coq'Art: The calculus of inductive construction, in: Text in Theoretical Computer Science, in: An EATCS Series, Springer-Verlag, 2004.
[4] Y. Bertrand, J.-F. Dufourd, Algebraic specification of a 3D-modeler based on hypermaps, Graphical Models and Image Processing 56 (1) (1994) 29–60.
[5] H.R. Brahana, Systems of circuits on two-dimensional manifold, Annals of Mathematics 23 (2) (1926) 144–168.
[6] T. Coquand, G. Huet, Constructions: A higher order proof system for mechanizing mathematics, in: EUROCAL, in: LNCS, vol. 203, Springer-Verlag, 1985.
[7] The Coq Team Development - LogiCal Project: The Coq Proof Assistant Reference Manual — Version 8.1, INRIA, France, 2006. http://coq.inria.fr/V8.1/refman.
[8] R. Cori, Un Code pour les Graphes Planaires et ses Applications, in: Astérisque, vol. 27, Société Math. de France, 1970.
[9] H.S.M. Coxeter, Regular Polytopes, Dover, 1973.
[10] C. Dehlinger, J.-F. Dufourd, Formalizing generalized maps in Coq, Theoretical Computer Science 323 (2004) 351–397.
[11] C. Dehlinger, J.-F. Dufourd, Formalizing the trading theorem in Coq, Theoretical Computer Science 323 (2004) 399–442.
[12] J.-F. Dufourd, Formal specification of topological subdivisions using hypermaps, Computer-Aided Design 23 (2) (1991) 99–115.
[13] J.-F. Dufourd, Algebras and formal specifications in geometric modelling, The Visual Computer 13 (1997) 61–72.
[14] J.-F. Dufourd, F. Puitg, Fonctional specification and prototyping with combinatorial oriented maps, Computational Geometry — Theory and Applications 16 (2000) 129–156.
[15] J.-F. Dufourd, Design and formal proof of a new optimal image segmentation program with hypermaps, Pattern Recognition 40 (2007) 2974–2993.
[16] J.-F. Dufourd, Hypermaps, Genus theorem and Euler formula, Users'contributions, Coq site, INRIA. http://coq.inria.fr, 2008.
[17] J. Edmonds, A combinatorial representation for polyhedral surfaces, Notices American Mathematical Society 7 (1960).
[18] D. Eppstein, The Geometry Junkyard — Nineteen Proofs of Euler's Formula: V-E+F=2. http://www.ics.uci.edu/eppstein/junkyard/euler/.
[19] E. Flato, et al., The design and implementation of planar maps in CGAL, The ACM Journal of Experimental Algorithmics 16 (2000). Also in LNCS 1668 (WAE'99), Springer-Verlag, pp. 154–168.
[20] M. Gangnet, J.C. Hervé, T. Pudet, Incremental computation of planar maps, in: ACM Siggraph Conf, Computer Graphics 23 (1989) 121–130.
[21] G. Gonthier, A computer-checked proof of the Four Colour Theorem, Microsoft Research, Cambridge. http://coq.inria.fr/doc/main.html, 2005, 57 pages.
[22] H. Griffiths, Surfaces, Cambridge University Press, 1981.
[23] T. Hales, A verified proof of the Jordan curve theorem, Seminar Talk. Dep. of Math., University of Toronto, 2005.
[24] G. Huet, G. Kahn, C. Paulin-Mohring, The Coq Proof Assistant — A Tutorial — Version 8.0, Tech. Report, INRIA, France, 2004. http://coq.inria.fr/doc/main.html.
[25] A. Jacques, Sur le genre d'une paire de substitutions, C. R. Acad. Sci. Paris 267 (18) (1968) 625–627. Série A.
[26] A. Jacques, Constellations et graphes topologiques, in: Combinatorial Theory and Applications Symp., Budapest, 1970, pp. 657–673.
[27] G. Kahn, Elements of Constructive Geometry, Group Theory, and Domain Theory. Coq contribution. http://coq.inria.fr/contribs-eng.html.
[28] D.E. Knuth, Axioms and Hulls, in: LNCS, vol. 606, Springer-Verlag, 1992.
[29] I. Lakatos, Proofs and Refutations, in: J. Worral, E. Zahar (Eds.), The Logic of Mathematical Discovery, Cambridge University Press, 1976.
[30] P. Letouzey, A new extraction for Coq, in: TYPES, in: LNCS, vol. 2646, Springer-Verlag, 2002, pp. 200–219.
[31] P. Lienhardt, Topological models for boundary representation — A survey, Computer Aided Design 23 (1991) 59–81.
[32] S. Lifschetz, The early development of algebraic topology, in: I.M. James (Ed.), History of Topology, Elsevier, 1999, pp. 531–560 (Chapter 18).
[33] M. Mäntylä, R. Sulonen, GWB: A solid modeler with Euler operators, IEEE Computer Graphics and Applications 2 (1984) 17–31.
[34] C. Paulin-Mohring, Inductive Definition in the System Coq — Rules and Properties, in: Typed Lambda-Calculi and Applications, vol. 664, Springer-Verlag, 1993, pp. 328–345.
[35] D. Pichardie, Y. Bertot, Formalizing convex hulls algorithms, in: Theorem Proving in HOL Conf., in: LNCS, vol. 2152, Springer-Verlag, 2001, pp. 346–361.
[36] J. von Plato, The axioms of constructive geometry, Annals of Pure and Applied Logic 76 (1995) 169–200.
[37] W. Pugh, The Omega Test: A fast and practical integer programming algorithm for dependence analysis, Communications of the ACM (1992) 102–114.
[38] F. Puitg, J.-F. Dufourd, Formal specifications and theorem proving breakthroughs in geometric modelling, in: Theorem Proving in HOL Conf., in: LNCS, vol. 1479, Springer-Verlag, 1998, pp. 401–427.
[39] F. Puitg, Preuves en modélisation géométrique par le calcul des constructions inductives, Ph.D. Thesis, Strasbourg University, 1999.

[40] F. Puitg, J.-F. Dufourd, Formalizing mathematics in higher-order logic: A case study in geometric modelling, Theoretical Computer Science 234 (2000) 1–57.
[41] A.A.G. Requicha, Representations for rigid solids, ACM Computing Surveys 12 (4) (1980) 437–464.
[42] P. Schorn, An axiomatic approach to robust geometric programs, Journal of Symbolic Computation 16 (1980) 155–165.
[43] S. Stahl, A combinatorial analog of the jordan curve theorem, Journal of Combinatorial Theory 26 (Serie B) (1983) 28–38.
[44] W.E. Tutte, Combinatorial oriented maps, Canadian Journal of Mathematics XXXI (5) (1979) 986–1004.
[45] W.E. Tutte, Graph theory, in: Encyclopedia of Mathematics and its Applications, Addison Wesley, Reading, MA, 1984.
[46] M. Yamamoto, S. Nishizaki, M. Hagiya, T. Tamai, Formalization of planar graphs, in: Theorem Proving in HOL Conf., in: LNCS, vol. 971, Springer-Verlag, 1995, pp. 369–384.